

PoseAgent: Budget-Constrained 6D Object Pose Estimation via Reinforcement Learning

Alexander Krull¹, Eric Brachmann¹, Sebastian Nowozin²,
Frank Michel¹, Jamie Shotton², Carsten Rother¹

¹ TU Dresden, ² Microsoft

Abstract

State-of-the-art computer vision algorithms often achieve efficiency by making discrete choices about which hypotheses to explore next. This allows allocation of computational resources to promising candidates, however, such decisions are non-differentiable. As a result, these algorithms are hard to train in an end-to-end fashion. In this work we propose to learn an efficient algorithm for the task of 6D object pose estimation. Our system optimizes the parameters of an existing state-of-the-art pose estimation system using reinforcement learning, where the pose estimation system now becomes the stochastic policy, parametrized by a CNN. Additionally, we present an efficient training algorithm that dramatically reduces computation time. We show empirically that our learned pose estimation procedure makes better use of limited resources and improves upon the state-of-the-art on challenging datasets. Our approach enables differentiable end-to-end training of complex algorithmic pipelines and learns to make optimal use of a given computational budget.

1. Introduction

Many tasks in computer vision involve *learning a function*, usually learning to predict a desired output label given an input image. Advances in deep learning have led to huge progress in solving such tasks. In particular, convolutional neural networks (CNNs) work well when trained over large training sets using gradient descent methods to minimize the expected loss between the predictions and the ground truth labels.

However, important computer vision algorithms take the form of *algorithms* rather than being a simple differentiable function: example sliding window search, superpixel partitioning, particle filters, and classification cascades are algorithms realizing complex non-continuous functions.

The *algorithmic approach* is especially useful in situa-

tions where computational budget is limited: an algorithm can dynamically assign its budget to solving different aspects of the problem, for example, to take shortcuts in order to spend computation on more promising solutions at the expense of less promising ones.

We would like to *learn the algorithm*. Unfortunately, the hard decisions taken in most algorithmic approaches are non-differentiable, and this means that the structure and parameters of these efficient algorithm cannot be easily learned from data.

Reinforcement learning (RL) [22] offers a possible solution to learning algorithms. We view the algorithm as the *policy* of an RL agent, *i.e.* a description of dynamic sequential behaviour. Then RL provides a framework to learn the parameters of such behaviour with the goal of maximizing an expected reward, for example, the accuracy of the algorithm output.

We apply this perspective on algorithmic computer vision approaches. In particular, we address the problem of 6D object pose estimation and use RL to learn the parameters of a deep algorithmic pipeline to provide the best possible accuracy given a limited computational budget.

Object pose estimation is the task of estimating from an image the 3D translation (position) and 3D rotation (orientation) of a specific object relative to its environment. This task is important in many applications such as robotics and augmented reality where the efficient use of a limited computation budget is an important requirement.

State-of-the-art pose systems such as the system of Krull *et al.* [12] generate a pool of pose hypotheses, then score each hypothesis using a pre-trained CNN. The subset of high-scoring hypotheses get refined and ultimately the highest-scoring hypothesis is returned as the answer. Computationally the refinement step is the most expensive, and there is a trade-off between the number of refinements allowed and the expected quality of the result.

Ideally, one would train such state-of-the-art system end-to-end in order to learn how to use the optimal number of refinements to maximize the expected success of pose

estimation. Unfortunately, treating the system as a black box with parameters to optimize is impossible for two reasons: (i) each selection process is non-differentiable with respect to the scoring function; and (ii) the loss used to determine whether an estimated pose is correct is also non-differentiable.

We recast pose estimation as an RL problem in order to overcome these difficulties. We model the pose inference process as an RL agent which we call *PoseAgent*. *PoseAgent* is granted more flexibility than the original system: it is given a fixed budget of refinement steps, and is allowed to manipulate its hypothesis pool by selecting individual poses for refinement, until the budget is spent.

In our *PoseAgent* model each agent decision follows a probability distribution over possible actions. This distribution is called the policy and we can differentiate and optimize this continuous policy through the stochastic policy gradient approach [23]. As a result of this stochastic approach the final pose estimate becomes a random variable, and each run of *PoseAgent* will produce a slightly different result.

This policy gradient approach is very general and does not require differentiability of the used loss function. As a consequence we can directly take the gradient with respect to the expected loss of interest, *i.e.* the number of correctly estimated poses.

Training in policy gradient methods can be difficult due to the additional variance of estimated gradients [7, 23], because the additional randomness leads to a bigger variance in the estimated gradients. To overcome this problem we propose an efficient training algorithm that radically reduces the variance during training compared to a naïve technique.

We compare our approach to the state-of-the-art [12] and achieve substantial improvements in accuracy, while using the same or smaller average budget of refinement steps compared to [12].

In summary our **contributions** are:

- To the best of our knowledge, we are the first to apply a policy gradient approach to the object pose estimation problem.
- Our approach allows the use of a non-differentiable reward function corresponding to the original evaluation criterion.
- We present an efficient training algorithm that dramatically reduces the variance during training.
- We improve significantly upon the best published results on the dataset.

2. Related Work

Below, we first discuss approaches for 6D pose estimation, focusing in particular on object coordinate prediction

methods, and then provide a short review of RL methods used in a setting similar to ours.

2.1. Pose Estimation

There is a large variety of approaches for 6D object pose estimation. Traditionally, approaches based on sparse features [14, 15] have been successful, but work well only for textured objects. Other approaches include template-based methods [9, 19], voting schemes [6, 10], and CNN-based direct pose regression [8].

We focus on the line of work called object coordinate regression [3], which provides the basic framework for our approach. Object coordinate regression was originally proposed for human body pose estimation [24] and camera localization [20]. In [3] a random forest provides a dense pixel-wise prediction for 6D object pose prediction. At each pixel, the forest predicts whether and where the pixel is located on the surface of the object. One can then efficiently generate pose hypotheses by sampling a small set of pixels and combining the forest predictions with depth information from an RGB-D camera.

The object coordinate regression methods in [3, 12, 17] score these hypotheses by comparing rendered and observed image patches. While [3, 17] use a simple pixel wise distance function, [12] propose a learned comparison: a CNN compares rendered and observed images and outputs an energy value representing a parameter of the posterior distribution in pose space. Despite their differences in the particular scoring functions, [3, 17, 12] use the same inference technique to arrive at the final pose estimate: they all refine the best hypotheses, re-score them, and output the best one as their final choice. Our *PoseAgent* approach can be seen as a generalization of this algorithm, in which the agent selects the hypotheses for refinement repeatedly, each time being able to make a more informed choice.

The work of Krull *et al.* [12] is the most closely related to our work. We use a similar CNN construction as Krull *et al.*, feeding both rendered and observed image patches into to our CNN. However, we use the output of the CNN as a parameter of the stochastic policy that controls the behaviour of our pose agent. Moreover, while the training process in [12] is seen as learning the posterior distribution, which is then maximized during testing using the fixed inference procedure, our training process instead modifies the behaviour of the agent directly in order to maximize the number of correctly estimated poses.

2.2. Reinforcement Learning in Similar Tasks

RL has traditionally been successful in areas like robotics [21], control [1], advertising, network routing, or playing games. While the application of RL seems natural for such cases where real agents and environments are involved, RL is increasingly being successfully applied in

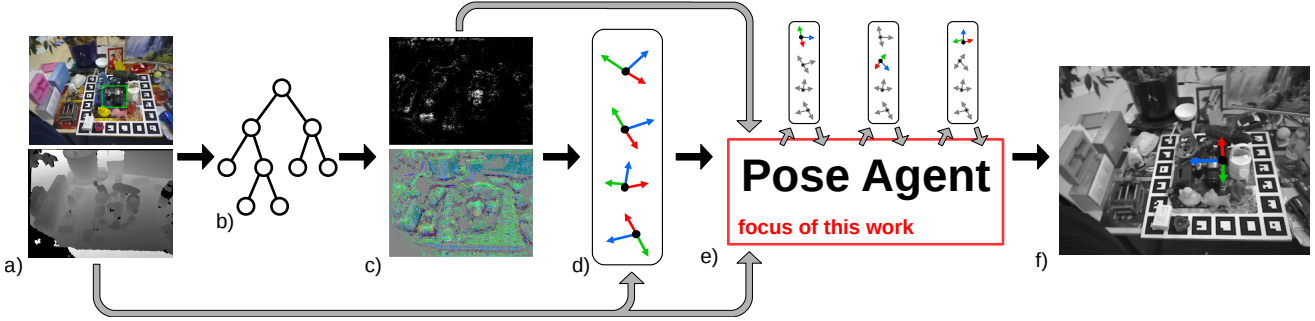


Figure 1. The pose estimation pipeline: a) The input of our system is an RGB-D image. We are interested in the pose of the camera highlighted by the green box. b) Similar to [3], the image is processed by a random forest. c) The forest outputs dense predictions of object probabilities (top) and object coordinates (bottom). The object coordinates are mapped to the RGB cube for visualization. d) We use the predictions together with the depth information to sample a pool of pose hypotheses H^0 . e) An RL agent manipulates the hypotheses pool by repeatedly selecting individual hypotheses to refine. This is the focus of this paper. f) The agent outputs a final pose estimate \tilde{H} .

computer vision systems where the interpretation of the system as an agent interacting with an environment is not always so intuitive. While we are to our knowledge the first to apply RL for 6D object pose estimation, there are several recent papers that apply RL for 2D object detection and recognition [18, 5, 16, 2].

In [18, 5], an agent shifts its area of attention over the image until it makes a final decision. Instead of moving a single 2D area of attention over search space like [18, 5], we work with a pool of multiple 6D pose hypotheses. The agent in [16] focuses its attention by moving a 2D fixation point, though operates on a set of precomputed image regions to gather information and make a final decision. Our agent instead manipulates its hypothesis pool by refining individual hypotheses.

Caicedo *et al.* [5] use Q-learning, in which the CNN predicts the quality of the available state-action pairs. Mnih *et al.* [18] and Mathe *et al.* [16] use a different RL approach based on stochastic policy gradient, in which the behaviour of the agent is directly learned to maximize an expected reward. We follow [18, 5] in using stochastic policy gradient, which allows us to use a non-differentiable reward function, directly corresponding to the final success criterion used during evaluation.

3. Method

In this section, we first define the pose estimation task and briefly review the pose estimation pipeline from [3, 12, 4]. We then continue to describe PoseAgent, our reinforcement learning agent, designed to solve the same problem. Finally, we discuss how to train our agent, introducing our new, efficient training algorithm.

3.1. Pose Estimation Pipeline

We begin by describing the object pose estimation task. Given an RGB-D image x we are interested in localizing a

specific, known, rigid object (Fig. 1a). We assume that exactly one object instance is present in the scene. Our goal is to estimate the true pose H^* of the instance, *i.e.* its position in space as well as its orientation. The pose has a total of six degrees of freedom, three for translation and three for rotation. We define the pose as the rigid body transformation that maps a point from the local coordinate system of the object to the coordinate system of the camera.

Our method is based on the work of Krull *et al.* [12]. As in [12], we use an intermediate image representation called object coordinates. By looking at small patches of the RGB-D input image, a random forest (Fig. 1b) provides two predictions for every pixel i . Each tree predicts an object probability $p_i \in [0, 1]$ as well as a set of object coordinates y_i (Fig. 1c). The object probability p_i describes whether the pixel is believed to be part of the object or not. The object coordinates y_i represent the predicted position of the pixel on the surface of the object, *i.e.* its 3D coordinates in the local coordinate system of the object.

Again following [12], we use these forest predictions in a RANSAC-inspired sampling scheme to generate pose hypotheses. We repeatedly sample three pixels from the image according to the object probabilities p_i . By combining the predicted object coordinate y_i with the camera coordinates of the pixels (calculated from the depth channel of the input image), we obtain three 3D-3D correspondences. We calculate a pose hypothesis from these correspondences using the Kabsch algorithm [11]. We sample a fixed number N of hypotheses, which are combined in hypothesis pool $H^0 = (H_1^0 \dots H_N^0)$ (Fig. 1d). The upper index denotes time steps which we will use later in our algorithm.

Krull *et al.* [12] proposed the following rigid scheme for pose optimization. All hypotheses are scored and the 25 top-scoring hypotheses are refined. Then, the refined hypotheses are scored again, and the best scoring hypothesis is returned as the final pose estimate of the algorithm.

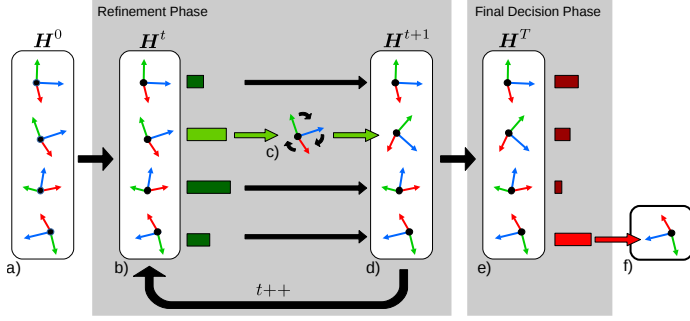


Figure 2. The pose agent inference process: a) The initial pool of hypotheses is sampled and handed to the agent. b) The agent selects a hypotheses $H_{a^t}^t$ by sampling from the policy $\pi(a^t|S^t; \theta)$. c) The selected hypothesis is refined. d) If refinement budget is left, the refinement phase continues. If the budget is exhausted a final selection is made. e) The final selection is made by sampling from the policy $\pi(a^T|S^T; \theta)$. f) The selected pose $H_{a^T}^T$ is output as final pose estimate.

Our paper focuses on improving the process by which the correct pose is found, starting from the same initial hypothesis pool. We propose to use an RL agent (Fig. 1e) to dynamically decide which hypothesis to refine next, in order to make most efficient use of a given computational budget. When its budget is exhausted, the agent selects a final pose estimate (Fig. 1f).

3.2. PoseAgent

We now describe our RL agent, PoseAgent, and how it performs inference. An overview of the process can be found in Fig. 3. The agent operates in two phases: (i) the refinement phase, in which the agent chooses individual hypotheses to undergo the expensive refinement step; and (ii) the final decision phase, in which it has to decide which pose should be selected as final output. In the following, we discuss both phases in detail.

Inference begins with the **refinement phase**. The pose agent starts with a pool $H^0 = (H_1^0 \dots H_N^0)$ of hypotheses which have been generated as described in Sec. 3.1, and a fixed budget B_0 of possible refinement steps.

At each time step t , the agent chooses one hypothesis index a^t , which we call an action. The chosen hypothesis is refined and the next time step begins. We limit the maximum number of times the agent may choose the same hypothesis for refinement to τ_{\max} . Hence, over time, the pool of actions (resp. hypotheses) the agent may choose for refinement decreases. We denote the set of possible actions $A^t = \{a \in \{1, \dots, N\} | \tau_a^t < \tau_{\max}\}$, where τ_a^t denotes how many times hypothesis a has been refined before time t . Subsequently, the agent modifies the hypothesis pool by refining hypothesis $H_{a^t}^{t+1} = g(H_{a^t}^t)$, where $g(\cdot)$ is the refinement function. All other hypotheses remain unchanged

$$H_{a^{t+1}}^{t+1} = H_{a^t}^t \forall a \neq a^t.$$

We perform refinement as follows (see also [12]). We render the object in pose $H_{a^t}^t$. Each pixel within the rendered mask is tested for being an inlier.¹ All inlier pixels are used to re-calculate the pose with the Kabsch algorithm. We repeat this procedure multiple times for the single, chosen hypothesis until the number of inlier pixels stops increasing or until the number m^t of executed refinement steps exceeds a maximum m_{\max} . The budget is decreased by the number of refinement steps performed, $B^{t+1} = B^t - m^t$. The agent proceeds choosing refinement actions until $B^t < m_{\max}$, in which case further refinement may exceed the total budget B^0 of refinement steps.

When this point has been reached, the refinement phase terminates, and the agent enters the **final decision phase** in which the agent chooses a hypothesis as the final output. We denote the final action as $a^T \in \{1 \dots N\}$ and the final pose estimate as $\tilde{H} = H_{a^T}^T$. The agent receives a reward of $r = 1$ in case the pose is correct or a negative reward of $r = -1$ otherwise. We use the pose correctness criterion from [9].

In the following, we describe how the agent makes its decisions. During both, the refinement phase and the final decision phase, the agent chooses from the hypothesis pool. We describe the agent behaviour by a probability distribution $\pi(a^t|S^t; \theta)$ referred to as "policy". Given the current state S^t , which contains information about the hypothesis pool and the input image x , the agent selects a hypothesis by drawing a sample from the policy. The vector θ of learnable parameters consists of CNN weights (described in Sec. 3.2.3). We will first give details on the state space S^t before explaining policy $\pi(a^t|S^t; \theta)$.

3.2.1 State Space

We model our state space in a way that allows us to use our new, efficient training algorithm, described in Sec. 3.3.1. We assume that the current state S^t of the hypothesis pool decomposes as $S^t = (s_1^t, \dots, s_N^t)$, where s_a^t will be called the state of hypothesis H_a^t . The state of an hypothesis contains the original input image x , the forest prediction z for the image, the pose hypothesis H_a^t , as well as a vector f_a^t of additional context features of the hypothesis (see Sec. 3.2.3). In summary, this gives $s_a^t = (x, z, H_a^t, f_a^t)$.

3.2.2 Policy

Our agent makes its decisions using a softmax policy. The probability of choosing a particular action a^t during the re-

¹A pixel i is tested for being an inlier for pose H by transforming its predicted object coordinates y_i to camera space using H . If the Euclidean distance between resulting camera coordinates and the observed coordinates at the pixel is below a threshold the pixel is considered an inlier.

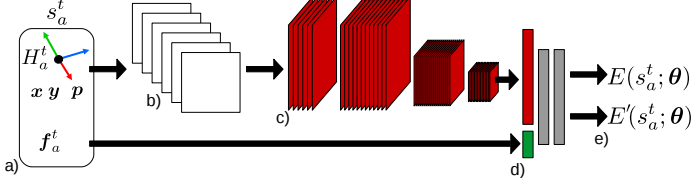


Figure 3. The CNN architecture: a) The system takes a pose hypothesis H_a^t and the additional features f_a^t encoding the context and history of the pose as input. b) We use the hypothesis to render the object and to cut out patches in the observed images. c) The images are processed by multiple convolutional layers. d) We concatenate the output of the convolutional layers with the features f_a^t . The result is fed into multiple fully-connected layers. e) The network predicts two energy values: $E(s_a^t)$ to be used in the refinement phase and $E'(s_a^t)$ to be used in the final decision.

finement phase is given by

$$\pi(a^t|S^t; \theta) = \frac{\exp(-E(s_a^t; \theta))}{\sum_{a \in A^t} \exp(-E(s_a^t; \theta))}, \quad (1)$$

where $E(s_a^t; \theta)$ will be called the energy of the state s_a^t . We will abbreviate it as $E_a^t = E(s_a^t; \theta)$. The energy of a state in the softmax policy is a measure of how desirable it is for the agent to refine the hypothesis. We use the same policy in the final decision phase, but with a different energy $E'(s_a^t; \theta)$ abbreviated by E'^t_a . We use a CNN to predict both energies, E_a^t and E'^t_a . In the next section, we discuss the CNN architecture and how it governs the behaviour of the agent.

3.2.3 CNN Architecture

We give an overview of the CNN architecture used in this work in Fig. 3. As in [12], the CNN compares rendered and observed images. We use the same six input channels as in [12], namely: the rendered depth channel, the observed depth channel, a rendered segmentation channel, the object probability channel, a depth mask, and a single channel holding the difference between object coordinates.

There are however two major differences in our CNN compared to the one used in [12], highlighted in Fig. 3. Firstly, while Krull *et al.* predict a single energy value of a pose, we jointly predict two separate energy values: one energy E_a^t for the refinement phase and one energy E'^t_a for the final decision phase.

Secondly, we input additional features to the network by concatenating them to the first fully connected layer. The features are: The number of times the hypothesis has already been selected for refinement. The distance the hypothesis has moved during its last refinement. The average distance of the hypothesis before refinement to all other hypotheses in the original pool.

3.3. Policy Gradient Training

We will now discuss the training procedure for our PoseAgent. First, we will give a general introduction of policy gradient training, and then apply the approach to the PoseAgent. Finally, we will introduce an efficient algorithm that greatly reduces variance during training and makes training feasible.

The goal of the training is the maximization of the expected reward $\mathbb{E}[R]$. This expected value depends on the environment as well as on the policy of our agent. In stochastic policy gradient methods one attempts to approximate the gradient with respect to the policy parameters θ . Note that since we are dealing with the expected value it becomes possible calculate derivatives, even if the reward function itself is non differentiable. We can write the derivative of the expected reward with respect to each parameter θ_j in θ as

$$\frac{\partial}{\partial \theta_j} \mathbb{E}[R] = \mathbb{E} \left[R \frac{\partial}{\partial \theta_j} \ln p(s^{1:T}, a^{1:T}; \theta) \right], \quad (2)$$

where $p(s^{1:T}, a^{1:T}; \theta)$ is the probability of a particular sequence of states $s^{1:T} = (s^1 \dots s^T)$ and actions $a^{1:T} = (a^1 \dots a^T)$ to occur. Because of the Markov property of the environment, it is possible to decompose the probability and rewrite it as

$$\frac{\partial}{\partial \theta_j} \mathbb{E}[R] = \mathbb{E} \left[R \sum_{t=0}^T \frac{\partial}{\partial \theta_j} \ln \pi(a^t|S^t; \theta) \right]. \quad (3)$$

Following the REINFORCE algorithm by Williams *et al.* [25], we approximate Eq. (3) using sampled sequences $(s_k^{1:T_k}, a_k^{1:T_k})$, generated by running the agent, as described in Sec. 3.2, on training images

$$\frac{\partial}{\partial \theta_j} \mathbb{E}[R] \approx \frac{1}{M} \sum_{k=1}^M R_k \sum_{t=0}^T \frac{\partial}{\partial \theta_j} \ln \pi(a_k^t|S_k^t; \theta). \quad (4)$$

Let us now consider the PoseAgent, which uses the softmax policy. In this case, we can calculate the derivative

$$\frac{\partial}{\partial \theta_j} \ln \pi(a^t|S^t; \theta) = \frac{\partial}{\partial \theta_j} E_a^t - \sum_{a \in A^t} \pi(a|S_k^t; \theta) \frac{\partial}{\partial \theta_j} E_a^t, \quad (5)$$

where the derivative $\frac{\partial}{\partial \theta_j} E_a^t$ of the energy can be calculated using the standard back propagation algorithm.

3.3.1 Efficient Training

We will now introduce our training algorithm (Algorithm 1), to dramatically reduce the variance of estimated gradients training. The basic idea is to make use of the special decomposable structure of the state space and our policy. The advantage of our algorithm compared to the naïve implementation is illustrated in Fig. 4.

To motivate the algorithm consider the following. Starting from a hypothesis pool $\mathbf{H}^0 = (H_1^0 \dots H_N^0)$, only a finite number of different hypothesis states $s_a^\tau | \tau \in \{0, \dots, \tau_{\max}\}$ can occur during a run of our PoseAgent. Here, $s_a^\tau = (H_a^\tau, \mathbf{f}_a^\tau)$ shall denote the state of hypothesis a after it has been refined τ times.

The algorithm pre-computes all possibly occurring hypothesis states s_a^τ , and predicts the corresponding energy values E_a^τ using the CNN. While this comes with some computational expense, it allows us to rapidly sample large numbers of sequences without having to re-evaluate the energy function.

To illustrate why this is possible, let us now reconsider Eq. (5). We can also write it as

$$\frac{\partial}{\partial \theta_j} \ln \pi(a^t | S^t; \theta) = \sum_{a \in A^t} \frac{\partial}{\partial \theta_j} E_a^t \frac{\partial}{\partial E_a^t} \ln \pi(a^t | S^t; \theta), \quad (6)$$

where

$$\frac{\partial}{\partial E_a^t} \ln \pi(a^t | S^t; \theta) = \begin{cases} 1 - \pi(a^t | S^t; \theta) & \text{if } a = a^t \\ -\pi(a^t | S^t; \theta) & \text{else} \end{cases}. \quad (7)$$

If we combine Eq. (6) and Eq. (4) we can rewrite the derivative of our expected reward as

$$\sum_{\tau=0}^{\tau_{\max}} \sum_{a=1}^N \frac{\partial}{\partial \theta_j} E_a^\tau \sum_{k=1}^M \sum_{t=1}^{T_k} \mathbb{1}(s_{a,k}^t = s_a^\tau) \frac{\partial}{\partial E_a^\tau} \ln \pi(a^t | S_k^t; \theta) r_k. \quad (8)$$

Here, $\mathbb{1}(s_{a,k}^t = s_a^\tau)$ is the indicator function. It has the value 1 whenever the sampled hypothesis state $s_{a,k}^t$ corresponds to the hypothesis state s_a^τ , and the value 0 in any other case.

Our algorithm works by first calculating the inner sums over k and t in Eq (8) for all τ and a . We compute these sums with a single iteration over all sequences k and all time steps t by accumulating the accordant summands in the entries $D(\tau_k^t, a)$ of table D . The accumulation of these values is computationally cheap, because it does not require any rendering or involvement of the CNN.

This structure allows us to increase the number of sampled sequences M without much computational cost. The algorithm can process an arbitrary amount of sequences using only a single back propagation pass of the CNN for each possible hypothesis state s_a^t . In a naive implementation, the number of required forward-backward passes would increase linearly with the number of sampled sequences.

Let us look at the algorithm in detail. It consists of three parts:

During the **Initialization Phase** we generated the original hypothesis pool as described in 3.1. Then, we refine all

of hypotheses τ_{\max} times and predict the energy values E_a^τ and $E_a^{\prime\tau}$ for all of them.

During the **Sampling Phase** we sample sequences $(s_k^{1:T}, a_k^{1:T})$ using the precomputed energies. We observe the reward r_k for each sequence and accumulate $\frac{\partial}{\partial E_a^\tau} \ln \pi(a^t | S_k^t; \theta) r_k$. This corresponds to the inner sums in Eq. (8) and has a very low computational cost per sequence.

During the final **Gradient Update Phase** we once more process each of the hypothesis states s_a^τ with the CNN and use the accumulated derivatives from the previous phase to update the derivatives of the CNN parameters θ . This corresponds to the outer sums in Eq. (8).

Input: image x , object coordinates y , object probabilities p

Initialization Phase:

generate hypothesis pool \mathbf{H}^0 ;
refine each hypotheses H_a^0 for τ_{\max} times and store resulting s_a^τ ;
calculate and store E_a^τ and $E_a^{\prime\tau}$;

Sampling Phase:

for all $k = 1 : K$ **do**
 sample path $(s_k^{1:T}, a_k^{1:T})$ using pre-computed E_a^τ and $E_a^{\prime\tau}$;
 receive reward r_k ;
 for all $t = 1 : T$ **do**
 for all $a \in A^t$ **do**
 calculate $\frac{\partial}{\partial E_a^\tau} \ln \pi(a^t | S_k^t; \theta) r_k$ and
 accumulate results in table entry $D_a^{\tau_{a,k}}$
 end
 end
end

Gradient Update Phase:

use accumulated results from table to apply back prop. for all s_a^τ ;

Algorithm 1: Efficient Training

4. Experiments

In the following we will describe the experiments to compare our method to the baseline system from [12]. Our experiments confirm, that our learned inference procedure is able to use its budget in a more efficient way, compared to the baseline from [12]. It outperforms the baseline system, while using on average a smaller number refinement steps.

Additionally we will describe an experiment regarding the efficiency of our training algorithm compared to a naïve

implementation of the REINFORCE algorithm. We find, that our algorithm can dramatically reduce the gradient variance during training.

We conducted our experiments on the **dataset** introduced in [13]. It features six RGB-D sequences of hand held sometimes strongly occluded objects.

We use the same division suggested by [12], reserving the sequences *samurai 1*, *cat 1* and *toolbox 1* for training and validation, while testing on the remaining *samurai 2*, *cat 2* and *toolbox 2* sequences.

4.1. Training and Validation Procedure

We train our system on the *samurai 1* sequence. As [12] we omit the first 400 frames in the training sequence to achieve a higher percentage of occluded images.

We train our system with two different parameter settings: Using a hypothesis pool size of $N = 210$, which is the setting used in [12], and a larger pool size of $N = 420$.

To determine the adequate size of the budget B of refinement steps, we ran the system from [12] on our validation set and determined the average number of refinement steps it used. We set our budget during training to the resulting number $B = 77$.

During training we allow a hypothesis to be chosen $\tau_{\max} = 3$ times for refinement. We set the maximum number of refinement steps per iteration to $m_{\max} = 10$.

Using stochastic gradient descent, we go through our training images in random order and run Algorithm 1 to approximate the gradient. We sample $M = 50k$ sequences for every image, and use an additional $50k$ to estimates a baseline b . We perform a parameter update after every image. Starting with an initial learning rate of $\lambda^0 = 25 \cdot 10^{-4}$, we reduce it according to $\lambda^l = \lambda^0 / (1 + l\nu)$, with $\nu = 0.01$. We use a fixed momentum of 0.9.

We skip all images in which none of the hypotheses from the pool leads to a correct pose after being refined τ_{\max} times. The reason for this is that such an image will always produce a reward of $r = -1$ and cannot provide meaningful information.

We also skip all images in which more than 10% of the hypotheses from the pool lead to a correct pose. Such images contribute little, because they are easily solved. We augment our training data, by randomly deciding for every image whether to mirror the CNN input along the x- and y-axis.

We run the training procedure for 96 hours on an Intel E5-2450 2.10GHz with Nvidia Tesla K20x GPU and save a snapshot every 50 training images. To avoid over-fitting, we test these saved snapshots on our validation set and choose the model with the highest accuracy.

In order to reduce the computational time during validation, we considered only images in which the object was at

least 5% occluded².

4.2. Testing Conditions

We compared both versions of our model, using $N = 210$ and $N = 420$, against the system of [12] using the corresponding pool size. In all experiments with the baseline method [12], we use the identical CNN with the original weights trained by Krull *et al.* in [12] on the *samurai 2* sequence. This is the network that [12] reports the best results for.

Apart from the pool size, we used the identical testing conditions as in [12], including the same random forest originally trained in [3]. To classify a pose in correct or false we use the same point-distance-based criterion used in [12]. A pose is considered correct, when the average distance between the vertices of the 3D model in the ground truth pose and the evaluated pose is below a threshold.

While the number of refinement steps in our setting is restricted, the method of [12] does not provide any guarantees on how many refinement steps are used. To ensure a fair comparison, we first ran the method from [12] and recorded the average number of refinement steps that it required on each test sequence. When running our method, we set the budget for each sequence to this recorded value, making sure that PoseAgent could never use more refinement steps than [12]. The total average number of refinement steps required by both methods can be found Table 1 and 2.

We evaluate our method using different parameters for τ_{\max} and m_{\max} , so that $\tau_{\max} \cdot m_{\max} \approx 30$. Meaning that a each pose can have an approximate maximum of 30 refinement steps. A higher value of τ_{\max} (and lower value of m_{\max}) means that PoseAgent can make more fine grained decisions on where to spend its budget. We use the following combinations for the two values ($\tau_{\max}=3, m_{\max} = 10$), ($\tau_{\max} = 5, m_{\max} = 6$), ($\tau_{\max} = 6, m_{\max} = 5$), ($\tau_{\max} = 7, m_{\max} = 4$).

4.3. Results

The results of our experiments can be seen in Table 1 and 2. PoseAgent is able to improve the best published results on the dataset by a total of **10.56%**. When we compare our method only to the baseline working on the same hypothesis pool size we are still able to outperform it. With the original pool size of $N = 210$ by **2.12%** and the increased pool size of $N = 420$ by **2.59%**.

Note, that the budget is set in a way, that is extremely restrictive, ensuring that PoseAgent can never use more refinement steps than [12] uses on average.

In both settings ($N = 210$ and $N = 420$) there appears to be a trend, that an increase of τ_{\max} , which corresponds to a more fine grained control of PoseAgent, leads

²according to the definition from [12]

Table 1. Percent correct poses using a hypothesis pool of $N = 210$

	Krull <i>et al.</i> [12]	Ours			
		$\tau_{\max}=3$	$\tau_{\max}=5$	$\tau_{\max}=6$	$\tau_{\max}=7$
Cat 2	63.05	67.81	68.61	71.52	68.74
Samurai 2	60.30	51.66	53.32	54.82	51.66
Toolbox 2	52.96	52.07	60.06	54.44	59.76
Total	60.06	58.94	61.47	62.18	60.88

Avg. ref.steps.	68.71	62.94	65.91	66.60	67.20
-----------------	-------	-------	-------	-------	-------

Table 2. Percent correct poses using a hypothesis pool of $N = 420$

	Krull <i>et al.</i> [12]	Ours			
		$\tau_{\max}=3$	$\tau_{\max}=5$	$\tau_{\max}=6$	$\tau_{\max}=7$
Cat 2	72.98	74.70	74.97	76.29	78.01
Samurai 2	66.45	59.30	59.63	58.80	61.13
Toolbox 2	64.79	65.38	68.93	71.60	71.01
Total	68.03	67.37	68.32	69.14	70.62

Avg. ref.steps.	71.12	65.00	68.04	68.66	69.39
-----------------	-------	-------	-------	-------	-------

to an improvement in accuracy. The only exception here is $\tau_{\max} = 7$ in the $N = 210$ setting. It should be noted that PoseAgent was trained with a different setting of $\tau_{\max} = 3$ and was able to generalize to the different settings used during testing.

4.4. Additional Experiments on the Efficiency of the Training Algorithm

In order to investigate the efficiency of our training algorithm compared to a naïve implementation of the REINFORCE algorithm, we conducted the following experiment:

We ran our training algorithm as well as the naïve implementation 100 times on a single training image without updating the network.

To estimate the variance of the gradient, we calculated the standard deviation of 1000 randomly selected elements from the resulting gradient vector of the CNN and averaged them. We recorded the required computation time to process the image on an Intel E5-2450 2.10GHz with Nvidia Tesla K20x GPU.

The process was repeated for $M = 5$, $M = 50$, $M = 500$, $M = 5000$ and $M = 50000$ sequences in case of the efficient algorithm. In case of the naïve implementation we used $M = 1$, $M = 2$, $M = 3$ and $M = 4$ sequences.

To keep the computation time in reasonable limits we used a reduced setting with a hypothesis pool size of $N = 21$ for both methods.

As can be seen in Fig. 4 our algorithm allows us to reduce variance greatly with almost no increase in computation time.

5. Conclusion

We have demonstrated a method learn the algorithmic inference procedure in a pose estimation system using a policy gradient method. Our system learns to make efficient

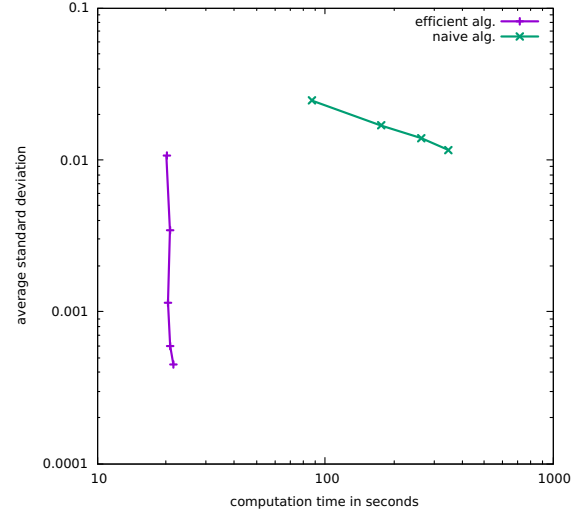


Figure 4. Observed gradient variance during training as a function of time: Our method is able process dramatically more sequences with almost no increase in computation time compared to a naïve implementation of the REINFORCE algorithm. The result is a drastically reduced gradient variance

use of a given budget and is able to outperform the original system, while using on average less computational resources. We have presented an efficient algorithm for the gradient approximation during training. The algorithm is able to sharply reduce gradient variance, without a significant increase in computation time.

We see multiple interesting future directions of research in the context of our system. (i) One could investigate a soft version of PoseAgent, which is not working with a fixed budget, but can instead decide what is the appropriate time to stop. In such systems the used computational budget can be part of the reward function. (ii) The sequential structure of the current system does not allow simple parallelization, but a PoseAgent that learns to do inference making use of multiple computational cores could be conceived.

5.0.1 Acknowledgements.

This work was supported by: European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 647769); German Federal Ministry of Education and Research (BMBF, 01IS14014A-D); EPSRC EP/I001107/2; ERC grant ERC- 2012-AdG 321162-HELIOS. The computations were performed on an HPC Cluster at the Center for Information Services and High Performance Computing (ZIH) at TU Dresden.

We thank Joachim Staib for his help with CPU rendering on the Cluster.

References

- [1] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19:1, 2007. [2](#)
- [2] J. Ba, V. Mnih, and K. Kavukcuoglu. Multiple object recognition with visual attention. In *ICLR'15*, 2015. [3](#)
- [3] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother. Learning 6d object pose estimation using 3d object coordinates. In *ECCV*, pages 536–551, 2014. [2](#), [3](#), [7](#)
- [4] E. Brachmann, F. Michel, A. Krull, M. Y. Yang, S. Gumhold, and C. Rother. Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image. In *CVPR*, 2015. [3](#)
- [5] J. C. Caicedo and S. Lazebnik. Active object localization with deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2488–2496, 2015. [3](#)
- [6] B. Drost, M. Ulrich, N. Navab, and S. Ilic. Model globally, match locally: Efficient and robust 3d object recognition. In *CVPR*, volume 1, page 5, 2010. [2](#)
- [7] E. Greensmith, P. L. Bartlett, and J. Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530, 2004. [2](#)
- [8] S. Gupta, P. Arbeláez, R. Girshick, and J. Malik. Inferring 3d object pose in RGB-D images. In *CVPR*, 2015. [2](#)
- [9] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *ACCV*, 2012. [2](#), [4](#)
- [10] S. Hinterstoisser, V. Lepetit, N. Rajkumar, and K. Konolige. Going further with point pair features. In *European Conference on Computer Vision*, pages 834–848. Springer, 2016. [2](#)
- [11] W. Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, 32(5):922–923, 1976. [3](#)
- [12] A. Krull, E. Brachmann, F. Michel, M. Ying Yang, S. Gumhold, and C. Rother. Learning analysis-by-synthesis for 6d pose estimation in rgb-d images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 954–962, 2015. [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#)
- [13] A. Krull, F. Michel, E. Brachmann, S. Gumhold, S. Ihrke, and C. Rother. 6-dof model based tracking via object coordinate regression. In *ACCV*, 2014. [7](#)
- [14] D. G. Lowe. Local feature view clustering for 3d object recognition. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–682. IEEE, 2001. [2](#)
- [15] M. Martinez Torres, A. Collet Romea, and S. Srinivasa. Moped: A scalable and low latency object recognition and pose estimation system. In *ICRA*, 2010. [2](#)
- [16] S. Mathe, A. Pirinen, and C. Sminchisescu. Reinforcement learning for visual object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2894–2902, 2016. [3](#)
- [17] F. Michel, A. Krull, E. Brachmann, M. Y. Yang, S. Gumhold, and C. Rother. Pose estimation of kinematic chain instances via object coordinate regression. In *Proc. British Machine Vision Conf*, pages 181–1, 2015. [2](#)
- [18] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*, pages 2204–2212, 2014. [3](#)
- [19] R. Rios-Cabrera and T. Tuytelaars. Discriminatively trained templates for 3d object detection: A real time scalable approach. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2048–2055, 2013. [2](#)
- [20] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. W. Fitzgibbon. Scene coordinate regression forests for camera relocation in RGB-D images. In *CVPR*, pages 2930–2937, 2013. [2](#)
- [21] W. D. Smart and L. P. Kaelbling. Effective reinforcement learning for mobile robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 4, pages 3404–3410. IEEE, 2002. [2](#)
- [22] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998. [1](#)
- [23] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063, 1999. [2](#)
- [24] J. Taylor, J. Shotton, T. Sharp, and A. Fitzgibbon. The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 103–110. IEEE, 2012. [2](#)
- [25] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. [5](#)